Семестр 4 (2019), занятие 6. TCP/IP неблокирующий «Эхо» сервер, клиент

Клиент (функция `repl`)

```
#define BUFLEN 1024

void repl(int fd) {
    char buf[BUFLEN];
    for(;;) {
        printf("> ");
        if(!fgets(buf, sizeof buf, stdin))
            return;
        if(writeRead(fd, buf) == -1)
            return;
    }
}
```

Клиент (функция `writeRead`)

```
int writeRead(int fd, const char *txt) {
    uint32_t len;
    char     buf[BUFLEN];

    len = strlen(txt) + 1;

    if(write(fd, &len, sizeof len) != sizeof len) {
        fprintf(stderr, "Write(length) error.\n");
        return -1;
    }

    if(write(fd, txt, len) != (ssize_t)len) {
        fprintf(stderr, "Write(text) error.\n");
        return -1;
    }

    if(read(fd, &len, sizeof len) != sizeof len) {
        fprintf(stderr, "Read(length) error.\n");
        return -1;
    }

    if(len > sizeof buf) {
        fprintf(stderr, "Big message error.\n");
        return -1;
    }

    if(read(fd, buf, len) != (ssize_t)len) {
        fprintf(stderr, "Read(text) error.\n");
        return -1;
    }

    puts(buf);

    return 0;
}
```

Сервер (структура `conn_t`)

```
typedef struct {
    int     fd,
            actv;
    size_t  icur,
            iall,
            ocur,
            oall;
    char    ibuf[BUFLEN],
            obuf[BUFLEN];
} conn_t;

#define canRead(c)  \
    ((c)->actv && (c)->iall > (c)->icur)
#define canWrite(c) \
    ((c)->actv && (c)->oall > (c)->ocur)
```

Сервер (чтение)

```
static const char *prefix = "Echo: ";

void readConn(conn_t *c) {
    uint32_t h;
    ssize_t  s;

    if(!canRead(c)) return;

    s = read(c->fd,
             c->ibuf + c->icur,
             c->iall - c->icur);

    if(s > 0)
        c->icur += s;
    else
        if(!s)
            goto stop;
        else
            if(s     == -1     &&
               errno != EAGAIN &&
               errno != EWOULDBLOCK) {
                fprintf(stderr, "Read error.\n");
                goto stop;
            }

    if(c->iall == c->icur) {
        if(c->iall == sizeof h) {
            memcpy(&h, c->ibuf, sizeof h);
            c->iall += h;

            if(c->iall + strlen(prefix) > BUFLEN) {
                fprintf(stderr, "Big message error.\n");
                goto stop;
            }
        }
        else {
            h       = strlen(prefix) + c->iall - sizeof h;
            c->oall = h + sizeof h;

            memcpy(c->obuf, &h, sizeof h);
            sprintf(c->obuf + sizeof h,
                    "%s%s",
                    prefix,
                    c->ibuf + sizeof h);
        }
    }

    return;

stop:
    if(shutdown(c->fd, 2) == -1)
        fprintf(stderr, "Shutdown error.\n");

    if(close(c->fd))
        fprintf(stderr, "Close error.\n");

    c->actv = 0;
}
```

Сервер (запись)

```
void writeConn(conn_t *c) {
    ssize_t  s;

    if(!canWrite(c)) return;

    s = write(c->fd,
              c->obuf + c->ocur,
              c->oall - c->ocur);

    if(s > 0)
        c->ocur += s;
    else
        if(!s)
            goto stop;
        else
            if(s     == -1     &&
               errno != EAGAIN &&
               errno != EWOULDBLOCK) {
                fprintf(stderr, "Write error.\n");
                goto stop;
            }

    if(c->oall == c->ocur) {
        c->icur = 0;
        c->iall = 4;
        c->ocur = 0;
        c->oall = 0;
    }
    return;

stop:
    if(shutdown(c->fd, 2) == -1)
        fprintf(stderr, "Shutdown error.\n");

    if(close(c->fd))
        fprintf(stderr, "Close error.\n");

    c->actv = 0;
}
```

Сервер (новое соединение)

```
void newConn(int ld, conn_t *cs, size_t ncs) {
    struct sockaddr_in addr;
    socklen_t          addrlen;
    int                fd;

    memset(&addr, 0, sizeof addr);
    addrlen = sizeof addr;

    fd = accept(ld,
                (struct sockaddr *)&addr,
                &addrlen);
    if(fd == -1) {
        if(fd    == -1     &&
           errno != EAGAIN &&
           errno != EWOULDBLOCK)
            fprintf(stderr, "Accept error.\n");
        return;
    }

    printConn(fd, &addr);

    if(fcntl(fd, F_SETFL, O_NONBLOCK) == -1) {
        fprintf(stderr, "Nonblock error.\n");
        goto stop;
    }

    if((size_t)fd >= ncs) {
        fprintf(stderr, "Storage limit error.\n");
        goto stop;
    }

    cs[fd].fd   = fd;
    cs[fd].actv = 1;
    cs[fd].icur = 0;
    cs[fd].iall = 4;
    cs[fd].ocur = 0;
    cs[fd].oall = 0;

    return;

stop:
    if(shutdown(fd, 2) == -1)
        fprintf(stderr, "Shutdown error.\n");

    if(close(fd))
        fprintf(stderr, "Close error.\n");
}

void printConn(int fd, struct sockaddr_in *addr) {
    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET,
              &addr->sin_addr,
              ip,
              sizeof ip);
    printf("%4d -New (%s %d).\n",
           fd,
           ip,
           addr->sin_port);
}
```

Сервер (системный вызов poll)

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);


struct pollfd {
    int   fd;
    short events;
    short revents;
};
```

Сервер (функция loop)

```
void loop(int           ld,
          struct pollfd *ps,
          conn_t        *cs,
          size_t         ncs) {
    nfds_t nps;
    size_t i;
    short  e;

    ps[0].fd = ld;
    ps[0].events = POLLIN;

    while(!quit) {
        for(nps = 1, i = 0; i < ncs; i++)
            if(cs[i].actv) {
                e = 0;

                if(canRead(&cs[i]))
                    e = POLLIN;

                if(canWrite(&cs[i]))
                    e = e ? e | POLLOUT : POLLOUT;

                if(e) {
                    ps[nps].fd     = cs[i].fd;
                    ps[nps].events = e;
                    nps++;
                }
            }

        switch(poll(ps, nps, 1 * 1000)) {
            case 0:
                puts("Nothing");
                break;

            case -1:
                if(errno != EINTR)
                    fprintf(stderr, "Poll error.\n");
                break;

            default:
                if(ps[0].revents & POLLIN)
                    newConn(ld, cs, ncs);

                for(i = 1; i < nps; i++) {
                    if(ps[i].revents & POLLIN)
                        readConn(&cs[ps[i].fd]);

                    if(ps[i].revents & POLLOUT)
                        writeConn(&cs[ps[i].fd]);
                }
        }
    }
}
```

Контрольная работа

./client
> 12 1078<Enter>

./server

длина строки (int)
= 8

$\xrightarrow{\hspace{10em}}$

строка (char[])
= { '1', '2', ' ', '1', '0', '7', '8', 0 }

$\xrightarrow{\hspace{10em}}$

Печать: 11
Печать: 12 1078
Разложение на простые множители:
12: 2 2 3
1078: 2 7 7 11

количество чисел (int)
= 2

$\xleftarrow{\hspace{10em}}$

1-е число (int)
= 12

$\xleftarrow{\hspace{10em}}$

длина массива (int)
= 3

$\xleftarrow{\hspace{10em}}$

массив (int[])
$= \{2, 2, 3\}$

$\xleftarrow{\hspace{10em}}$

2-е число (int)
= 1078

$\xleftarrow{\hspace{10em}}$

длина массива (int)
= 4

$\xleftarrow{\hspace{10em}}$

массив (int[])
$= \{2, 7, 7, 11\}$

$\xleftarrow{\hspace{10em}}$

Печать: 2
Печать: 12: 2 2 3
Печать: 1078: 2 7 7 11
> ...
...
><Ctrl-D>